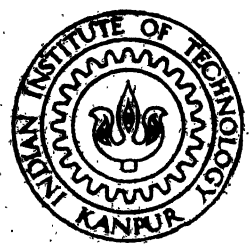
ANALYSING SORTING NETWORKS

by
MURALI KRISHNAN. K



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1998

Analysing Sorting Networks

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

Murali Krishnan K

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

April 1998

CERTIFICATE

This is to certify that the work contained in the thesis entitled Analysing Sorting Networks by Murali Krishnan K has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Manindra Agrawal,
Assistant Professor,
Department of Computer Science & Engineering,
Indian Institute of Technology, Kanpur.

Acknowledgment

I am indebted to a lot of people who loved me and cared for me during my stay in this campus.

My guide was to me a friend, teacher and constant source of inspiration, encouraging me in all my ventures, the culmination of all those this thesis is.

And then beyond academics, a group of lovers of Indian music who patiently groomed my little talents. You are the ones I will miss the most.

My friends - the ones here who helped me to discover myself and a few loving hearts far away.

And my parents, who guided me to be a more practical man in life.

I owe to you much much more than a few words, yet, Thank You!

Abstract

The study of Sorting Networks has been interesting owing to its various applications and rich underlying theory. In this thesis, we try to analyze sorting networks combinatorially using a new method based on **minimal sets**. A non-trivial lower bound obtained through arguments based on minimal sets is also presented.

Contents

List of Figures

Chapter 1

Introduction

In this chapter, we give an informal introduction to *comparison networks* and *sorting networks*, their representation, fundamental properties and applications.

1.1 Comparison Networks and their Representation

In this thesis, we study sorting networks, which are special forms of comparison networks. A comparison network is comprised solely of *wires* and *comparators*. A comparator (Fig.1(a)) is a device which takes two real valued inputs, x and y and produce two output values x' and y' according to the rule: $x' = \min(x, y)$, $y' = \max(x, y)$.

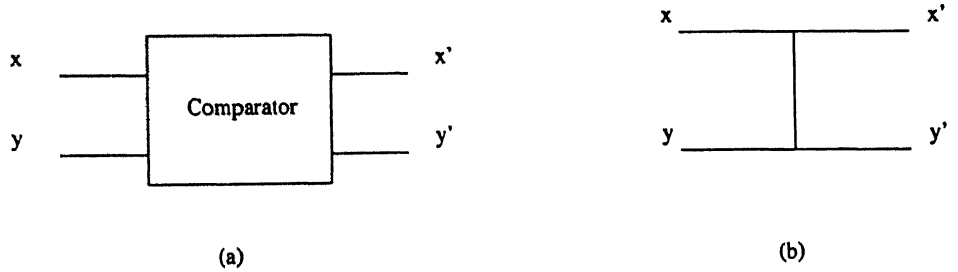


Figure 1: comparator

The compact representation of Fig.1(b) is often preferred and will be followed hereafter. A comparator thus sorts its two input values. We assume that a comparator operates in O(1) time.

A wire transmits a value from place to place. A wire can connect the output of one comparator to the input of another.

A *comparison network* is a set of comparators interconnected with wires.

Throughout this thesis, we assume that a comparison network contains n input wires through which the values $< x_1,x_2,..,x_n >$ to be processed by the network enter. The network also has n output lines in which the processed output values $< y_1,y_2,..,y_n >$ appear. We consider the input and the output as sequences $< x_1,..,x_n >$ and $< y_1,..,y_n >$ respectively. Fig.2 shows a comparison network.

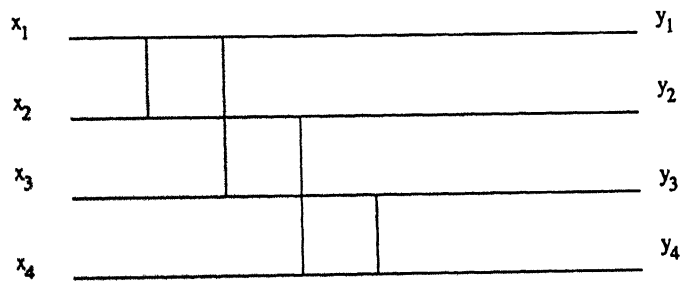


Figure 2: Comparison Network

We draw a comparison network as a collection of n horizontal lines with the comparators stretched vertically. A line actually does not represent a single wire, but a sequence of wires connecting various comparators. However, we follow the popular terminology and may use the term 'line' to indicate a wire when the meaning is evident from the context. Each comparator's input is connected to a wire that is either one of the networks' n input wires, or is connected to the output of another comparator. We consider only those networks which are not cyclic. ie., if we trace from the output of a given comparator to the input of another and so on, we shall never go through the same comparator twice. By convention, the network is drawn with inputs on the left and outputs on the right side.

1.2 Functioning of Comparison Networks

We assume that each comparator takes unit time for comparison, whereas, a wire can carry its input instantaneously from one end to another. Hence, if we trace any line, the total *delay* associated with the path traced is the number of comparators found in the line. *Running time* or *delay* of a comparison network is defined as the time it takes for all output wires to receive their values, once all the input wires receive theirs. Here and always, we assume that the inputs are applied simultaneously. Informally speaking, this time is the largest number of comparators that any input element can pass through as it travels from an input wire to an output wire. More formally, we define *depth* of a wire as follows. An input wire of a comparison network has depth 0. Now, if a comparator has two input wires with depth d_x and d_y, then its output wires have depth $\max(d_x, d_y)+1$. Since we do not allow cycles of comparators in any comparison network, the notion of depth becomes well defined. the depth or *level* of a comparator is defined as the depth of its output lines. The phrases like "comparators of the ith level" indicate the set of comparators in a comparison network having depth equal to i.

The depth of a comparison network is the maximum depth of an output wire in the network. The comparison network of Fig.2 for example has depth 3. We also define *size* of a comparison network as the total number of comparators in the network. The network of Fig.2 has size four.

Since a comparator is a passive element in the sense that it can only transpose its two inputs and not transform the values, a comparison network, which is just an interconnection of comparators is equally incapable of transforming values. Hence, we have the important observation that the **output sequence of a comparison network is a permutation of the input sequence**. Actually, much more can be said about comparison networks in general. For instance, during the above discussion, we have assumed that the inputs to the network are real valued. However, the general results about comparison networks can be seen to hold for input sequences from any ordered set with total order $\leq$ defined. This is owing to the intrinsic nature of the basic element, comparator, which is too weak for arithmetic and can only identify the relation $\leq$, and also due to the fact that our analysis (and interest)

is focused only on the ordering of elements at the output. Hence, we can as well consider sequences of integers (or from any ordered set) as inputs for the purpose of analysis. But the surprising fact that we can even do with much less, ie., with only input sequences from the set $\{0, 1\}$, with the normal partial order $\leq$ can be seen from Bouriciu's theorem. This and some more fundamental results on comparison and sorting networks will be discussed in the next chapter.

A comprehensive introduction into the subject can be seen in [1] where comparison networks are introduced in the context of the general problem of sorting. [2] also gives a simple introduction. A more formal development of the above concepts, treating comparison networks as a subclass of the more general communication networks can be seen in [3].

1.3 Sorting Networks

A sorting network is a comparison network whose output sequence is monotonically increasing ie., $y_1 \leq y_2 ... \leq y_n$ for *every* input sequence. Sorting networks form an important subclass of comparison networks.

Sorting networks are interesting because they afford an easy parallel hardware implementation. They also are interesting from a theoretical point of view since bounds on the complexity of several problems in general purpose communication networks, sorting, and boolean circuit complexity follow from the results on sorting networks, especially, from the celebrated Ajtai-Komlos-Szemeredi theorem [4], [5]. A brief account on the consequences of the AKS theorem can be seen in [3] which also presents a proof of the theorem. An account of the applications of sorting networks can be found in [8].

In this thesis, we will be centering our discussion on sorting networks, algorithms for their construction and complexity of the problem. In the next chapter, we develop some fundamental theory of sorting networks, some of which generalize to comparison networks. Also, a survey of the important problems and results are presented. This is followed by a chapter that presents our work on sorting networks, conclusions from our work and discussions on further research.

Chapter 2

A survey on Sorting Networks

This chapter is divided into three sections. In the first section, we shall look at some fundamental theorems on sorting networks. In the next section, we survey the developments in the problem of construction of efficient sorting networks and related subproblems. In the last section, we discuss results on the complexity of the problem of building efficient sorting networks.

2.1 Elementary Theory of Sorting Networks

Information theory asserts that, to sort n items, we need at least $\lg n!$ comparators. Hence, an n input sorting network (which will be called an n-sorter hereafter) requires at least $\lg n! = n\lg n - n\lg e - o(n)$ comparators. Since at a particular level (or depth) in a comparator network, we can have at most $n/2$ comparators, it follows that a depth of at least $(n\lg n - n\lg e - o(n))/(n/2) = 2\lg n - 2\lg e - o(1)$ is required for an n input comparison network to become an n-sorter. We shall come across better bounds later in this chapter.

It is quite non-trivial to prove that a given n input comparison network is an n-sorter for arbitrary n. No general efficient procedure is known. It would be sufficient to test with all $n!$ permutations of the sequence $< 1, 2, .., n >$ at the input. This fact follows from the observations we made in section 1.2. The following results show that in fact we can get by with far fewer tests.

2.1.1 Bouriciu's Theorem

If $f(x)$ is any *monotonic* function with $f(x) \leq f(y)$ whenever $x \leq y$ and if a given comparison network transforms the input sequence $x =< x_1, x_2, .., x_n >$ into the output sequence $y =< y_1, y_2, .., y_n >$, then the network will transform the input sequence $f(x) =< f(x_1), .., f(x_n) >$ into $f(y) =< f(y_1), .., f(y_n) >$.

Proof: We use induction to prove that if any wire in a comparison network assumes value x_i when the input sequence $< x_1, .., x_n >$ is applied, it assumes value $f(x_i)$ when input sequence $f(x) =< f(x_1), .., f(x_n) >$ is applied. Since the output lines are also included in the statement, proving this proves Bouriciu's theorem.

For wires at depth 0, viz., the input wires, the result follows trivially. For induction, consider a wire at depth d, $d \geq 1$. The wire is the output of a comparator at depth d and the input wires to this comparator are at a depth strictly less than d. By the inductive hypothesis, therefore, if the input wires to the comparator carry values x_i and x_j when input sequence x is applied, then they carry $f(x_i)$ and $f(x_j)$ when input sequence $f(x)$ is applied. The output wires of this comparator then carry $\min(f(x_i), f(x_j))$ and $\max(f(x_i), f(x_j))$ respectively. Now, since f is monotonically increasing, for every i, j $x_i \leq x_j$ implies $f(x_i) \leq f(x_j)$. Consequently, we have the identities:

$$\min(f(x_i), f(x_j)) = f(\min(x_i, x_j)); \max(f(x_i), f(x_j)) = f(\max(x_i, x_j)).$$

Thus, the upper and lower output wires of the comparator at depth d must have values $f(\min(x_i, x_j))$ and $f(\min(x_i, x_j))$ respectively. Since they carry $\min(x_i, x_j)$ and $\max(x_i, x_j)$ when input x is applied (this follows from the very definition of a comparator), the theorem is proved. **Q.E.D**

Now if the comparison network is a sorting network, we have the following remarkable result

2.1.2 Zero One Principle

If a comparison network with n input lines sorts all 2^n sequences from the set $\{0, 1\}^n$ correctly, then it sorts all sequences of arbitrary numbers correctly.

Proof: Suppose for the purpose of contradiction that the network sorts all zero-one sequences, but there exists a sequence $< x_1, ..x_n >$ which the network does not

sort, ie., there exists some x_i, x_j such that $x_i < x_j$, but the network places x_j before x_i in the output sequence. We define a monotonically increasing function, f as:

$$f(x) = \begin{cases} 0 & \text{if } x \leq x_i \\ 1 & \text{otherwise} \end{cases}$$

Since the network places x_j before x_i, by the statement which we have proved to yield the Bouriciu's theorem, the network places $f(x_j)$ before $f(x_i)$ in the output sequence when $< f(x_1, ..f(x_n) >$ is the input sequence. But since $f(x_j) = 1$ and $f(x_i) = 0$, we obtain the contradiction that the network fails to sort the zero-one sequence $< f(x_1, .., f(x_n) >$ correctly. Hence, the theorem is proved. **Q.E.D**

In our analysis of sorting networks, we use zero-one principle extensively and assume that inputs to the network are always zero-one sequences or vectors from $\{0,1\}^n$. By the above result it follow that results proved about sorting properties for such sequences will generalize for arbitrary input sequences from any ordered set.

2.1.3 Non Standard Models

Apart from the standard comparison networks seen so far, we also have non standard models which can have two types of comparators - one the standard one we have seen so far, and the non standard one which essentially functions in a similar way as the standard one, but with the output lines interchanged. ie., its top output line carrying the larger of the inputs and the bottom one carrying the smaller input. Since we need to differentiate between standard and non standard comparators, they are represented as shown in Figure 3. Figure 4 shows a non standard comparison network.

Knuth ([1], exercise 16, p239) has shown that any non standard network can be transformed into a standard one without increasing its depth or size. He has also given an algorithm for such transformation. We shall not discuss non standard models any further and so, we will view comparators in the conventional way.

In the next section, we shall survey the various methods for efficiently constructing sorting networks.

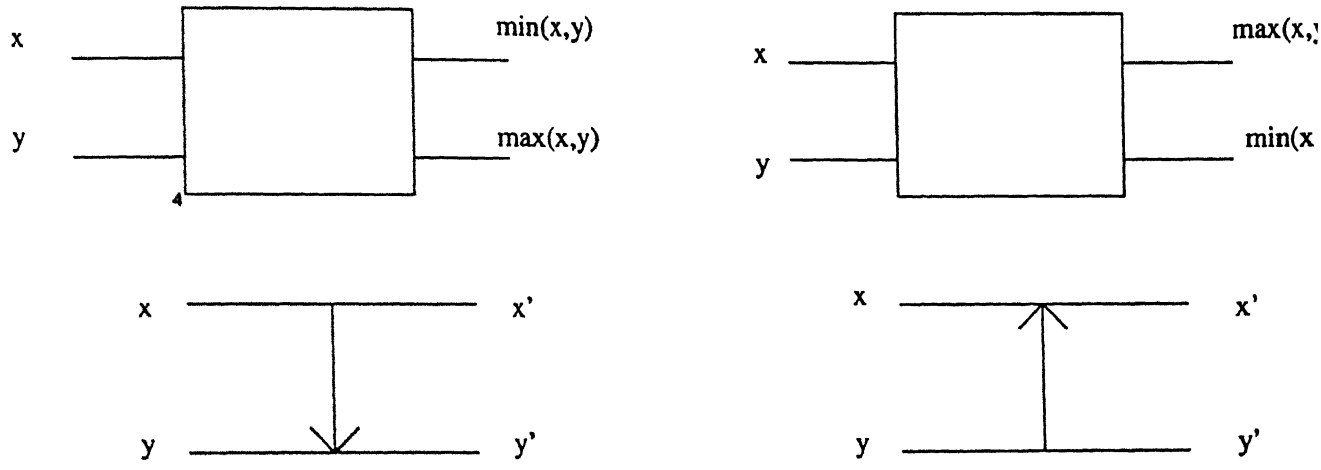


Figure 3: Standard and Non-Standard comparators

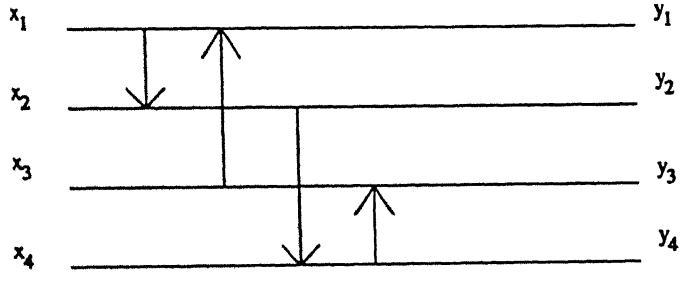


Figure 4: Non-Standard Comparison Network

2.2 Sorting Network Constructions

First, we shall discuss the straight forward methods for constructing sorting networks. This will be followed by a digression to discuss selection and merging networks. Finally, we shall take up the AKS theorem and its consequences.

2.2.1 Elementary Methods

The most natural way of constructing a sorting network for $n+1$ elements is to apply either the principle of *insertion* or the principle of *selection* on an n element

sorting network and continue recursively. Figure 5(a) shows how the $n+1^{th}$ element can be inserted into a sorted set of n lines. Fig 5(b) shows how to select the largest in $n+1$ lines and transfer it to the $n+1^{th}$ line.

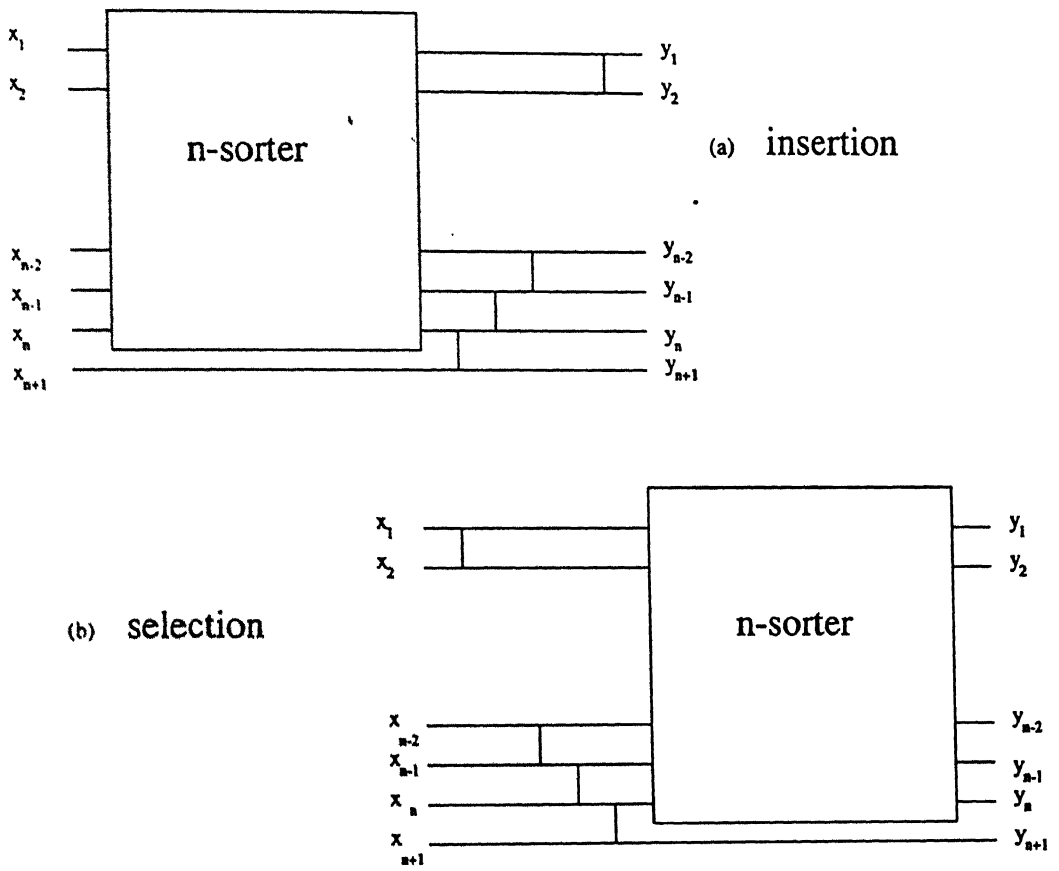


Figure 5: Making (n+1)-sorters from n-sorters

It is interesting to see that both methods reduce to the same "triangular" $2n-3$ delay procedure with $n(n+1)/2$ comparators. A 6-sorter constructed this way is shown below (Figure 6).

However, it is natural to think of more efficient constructions of depth $O(\log n)$ and hence size $O(n \log n)$ (the size bound occurring because at each level we can have at most $n/2$ comparators). This seems plausible in view of the information theoretic lower bound of $O(\log n)$ on depth.

However, the problem turned out to be unexpectedly hard until it was solved

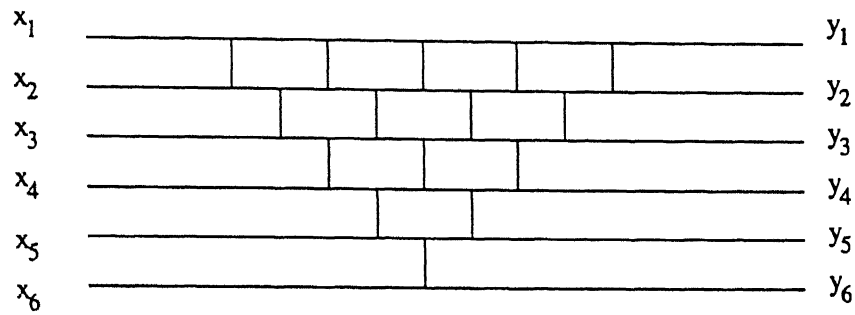


Figure 6: A 6-Sorter using insertion/selection

finally in 1983 by Ajtai, Komlos and Szemeredi (AKS). The rest of this section surveys the developments in the field leading to the celebrated AKS construction.

To yield bounds better than the straight sorting methods described above, attempts were made to *divide and conquer* the sorting problem with the help of *merging* and *selection* networks.

An (m, n)merging network (or (m, n)merger) takes two sets of inputs, each of m and n lines respectively, has $m + n$ output lines and has the property that if each of the input sets is sorted, then the output also is sorted (Fig 7(b)).

An (n, t)selection network (or (n, t)classifying network or (n, t)selector) is a network with n input lines and n output lines having the property that the t smallest inputs will be placed in the first t output lines (Fig 7(a)). Clearly, the definition is equivalent to saying that the $n - t$ largest inputs will be placed in the $n - t$ bottom lines. Hence, an (n, t)selector is also an $(n, n - t)$selector.

From this point on, we use the following notational conventions. $S_m(m, n)$ and $T_m(m, n)$ will denote the size and depth respectively of an (m, n)merger and $S_l(m, n)$ and $T_l(m, n)$ will denote those of an (m, n)selector under consideration. Similarly, $S_s(n)$ and $T_s(n)$ will denote the corresponding parameters for an n-sorter. The suffixes shall be dropped when the meaning is clear from the context.

Now, we can construct an n-sorter by merging the outputs from two $n/2$ sorters. Remembering the fact that a comparator is a $(1, 1)$merger, the procedure can be applied recursively (Fig 8(b)). In a similar way, since a comparator is also a (2,1)selector, we can use a (n,$n/2$)selector followed by two recursive $n/2$-sorters to produce

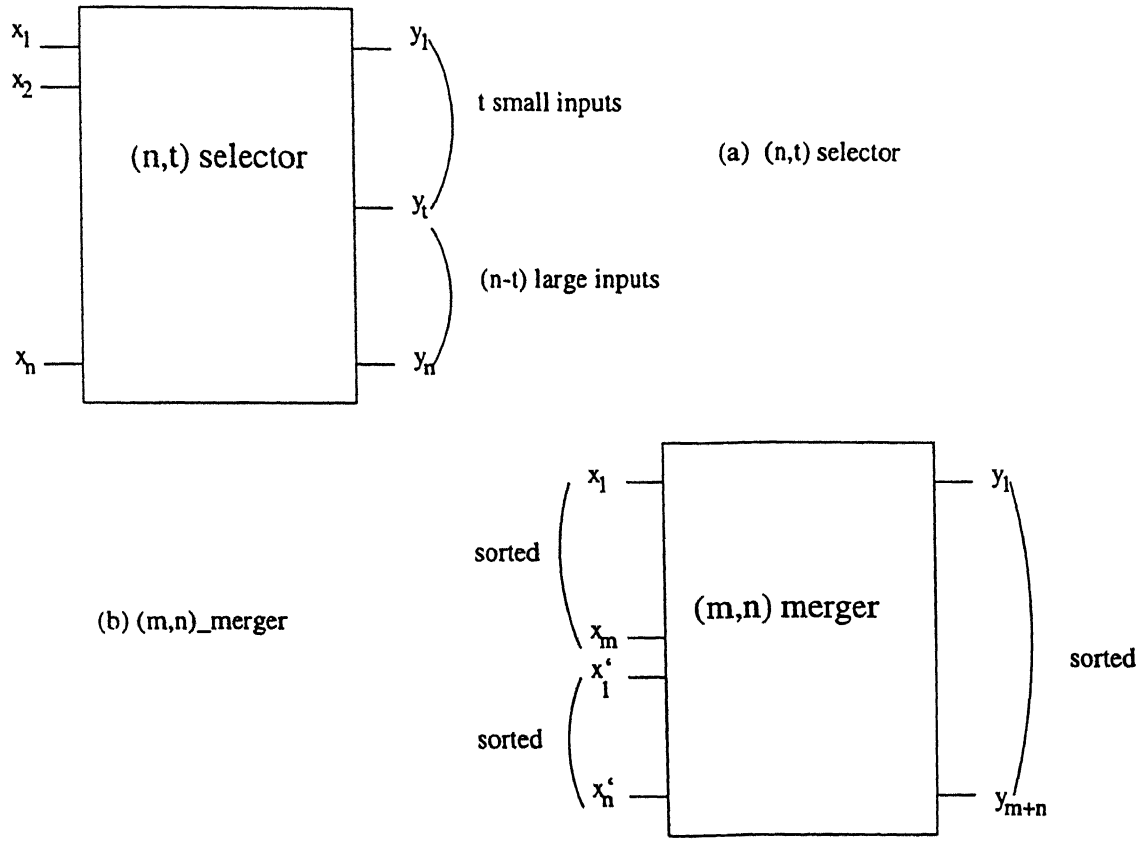


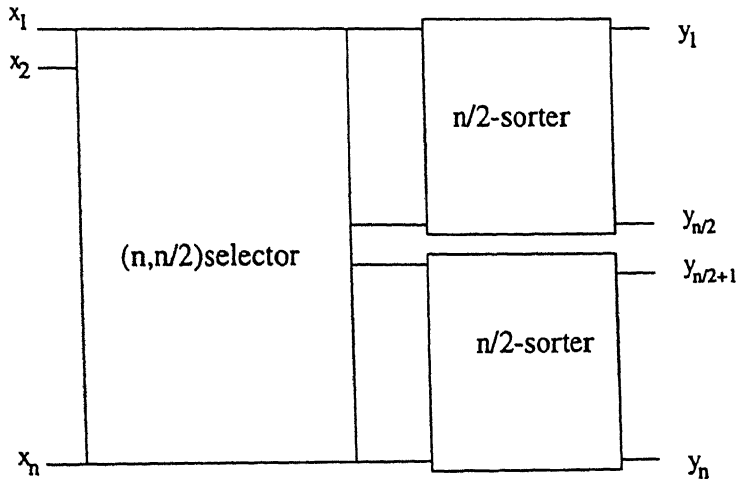
Figure 7: Schematic of Merge and Selection Networks

an n-sorter (Fig 8(a)). In both the cases, we get the following recurrences for size and depth respectively:

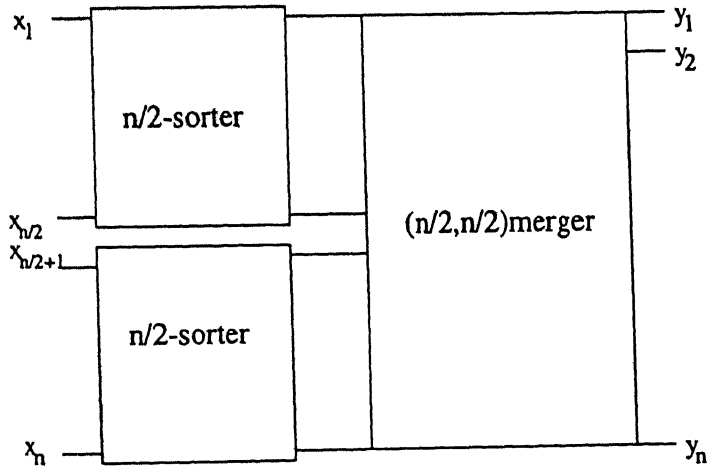
$$S_s(n) = 2S_s(n/2) + \begin{cases} S_m(n/2, n/2) & \text{for merge} \\ S_l(n, n/2) & \text{for selection} \end{cases}$$

$$T_s(n) = T_s(n/2) + \begin{cases} T_m(n/2, n/2) & \text{for merge} \\ T_l(n, n/2) & \text{for selection} \end{cases}$$

This reduces the focus to efficient construction of merging and selection networks, for, from the equation, it is clear that to produce $O(\log n)$ depth (and hence $O(n \log n)$ size) n-sorters, it would suffice to construct $O(1)$ delay merge or selection networks. However, the following discussion shows that this unfortunately is not possible.



(a)



(b)

Figure 8: Sorting Networks using: (a)selection, (b)merge.

2.2.2 Merge and Selection Networks

K.E.Batcher in 1968 devised a parallel merge exchange sorting scheme [6] which could be applied to construct sorting networks as well to give $O(\log^2 n)$ sorting network. A generalized version of the same can also be seen in [1](pp 224-227). The (n,n)merger used here has size $n \lg n + O(n)$. Another merge-sort network based on bitonic sorting is described in [2], again taking $O(\log^2 n)$ delay. Several minor improvements were made in this network, but all (n,n)merging schemes yet required $O(\log n)$ time and thus gave $O(\log^2 n)$ sorting networks.

Investigations for $O(1)$ delay (n,n)mergers were blocked when R.W.Floyd proved that $S_m(n,n) \geq \frac{1}{2} n \lg n + O(n)$ (and hence $T_m(n,n) \geq \lg n + O(1)$) giving an elegant simple proof ([1],pp 230-231). Later, A.C.Yao and F.F.Yao obtained tight lower bound for the merging problem [6] showing that $S_m(m,n) \geq n(\lg(m+1))/2$. The paper also showed that Batcher's parallel merge sort method was optimal and Floyd's bound was tight up to a constant factor.

V.E.Alekseyev had derived in 1969 ([1],p 234) an upper bound and a lower bound for (n,t)selectors. He showed the upper bound $S_l(n,t) \leq (n-t)(2S_s(t)/t+1)$, where $S_s(t)$ is the number of comparators required for a t-sorter. He also derived a lower bound $S_l(n,t) \geq (n-t)\lg\lceil t+1 \rceil$ using an ingenious proof. Later A.C.Yao [7] improved the lower bound for t-selector showing that $T_l(t,n) \geq \lg n + \lg\left(1/(t\lg\lceil t+1\rceil)\begin{pmatrix} T_l(t,n) \\ \lfloor \lg t \rfloor \end{pmatrix}\right)$. He also determined the asymptotic behavior of $T_l(n,t)$ as $n \to \infty$ to be $\lg n + \lfloor \lg t \rfloor \lg\lg n + O(\log\log\log n)$.

Thus, we see that $(n,n/2)$merger and $(n$,n/2)selector, both require $O(\log n)$ depth and a recursive construction of $O(\log n)$ n-sorter using either of them is not possible. At the beginning of the 1980's, it was generally believed that $O(\log n)$ delay n-sorter is an impossibility. Knuth actually went to the extent of presenting this open problem as "prove that the asymptotic value of S(n) is not $O(n \log n)$" ([1], problem 51, p 243).

However, contradictory to the general belief, Ajtai, Komlos and Szemeredi (AKS) came up with a proof for existence of n-sorters of depth $O(\log n)$ (and size $O(n \log n)$) [4][5]. Their proof is quite involved and uses properties of expander graphs. We will

not expound the proof here. For the proof, we refer to Pipinger [3]. However, we shall look at the fundamental principles in the construction of the AKS network.

2.2.3 The AKS Network

The key discovery of AKS was the fact that although selection (or classification in the AKS terminology) is an $\Omega(\log n)$ delay problem, one could do *approximate selection* or *approximate classification* in $O(1)$ time. An (n, t, ϵ)approximate classifier $(0 \leq \epsilon \leq 1)$ (or an (n, t)approximate classifier with tolerance ϵ) is a network which takes n input values and has the property that for all $1 \leq x \leq t$, at most ϵx of the x smallest elements are in the last $n - t$ lines of the output and for all $1 \leq x \leq n-t$ at most ϵx of the x largest elements are in the first t lines of the output. informally, this means that an approximate classifier with tolerance ϵ allows a small fraction ϵ of error. In particular, when $\epsilon = 0$, an (n, t, ϵ) approximate classifier becomes an (n, t)selector. Now, Ajtai, Komlos and Szmeredi discovered that for all $\epsilon > 0$ there exists a constant c such that, for every n there is a network that is an $(n, n/2, \epsilon)$approximate classifier of depth c (and size at most $cn/2$). AKS gave an explicit construction for such networks using expanders. They then came up with a construction for an n-sorter using $O(\log n)$ stages of approximate classifiers, thus yielding $O(\log n)$ sorting network and thereby solving an important open problem.

In the AKS network too, the sorting process proceeds recursively as in the case with the sorting networks based on $(n, n/2)$selectors. But since in this case we have a small error fraction, some sort of error correction mechanism is required. We can think of the sorting process as a binary tree. Initially all the "elements" to be sorted are at the root (Fig.9). In $O(1)$ time, these elements are approximately classified and send to the child nodes N_1 and N_2 and the process carries on. However, at each node the elements are further approximately classified (again in $O(1)$ time) and a few "large" elements from the left child and a few "small" elements from the right child are send back to the parent. This is because the erroneous elements are present among these extreme elements with high probability. Thus, at each node we have two stages of processing. But this can only increase the delay by a constant factor since each stage is of delay only $O(1)$.

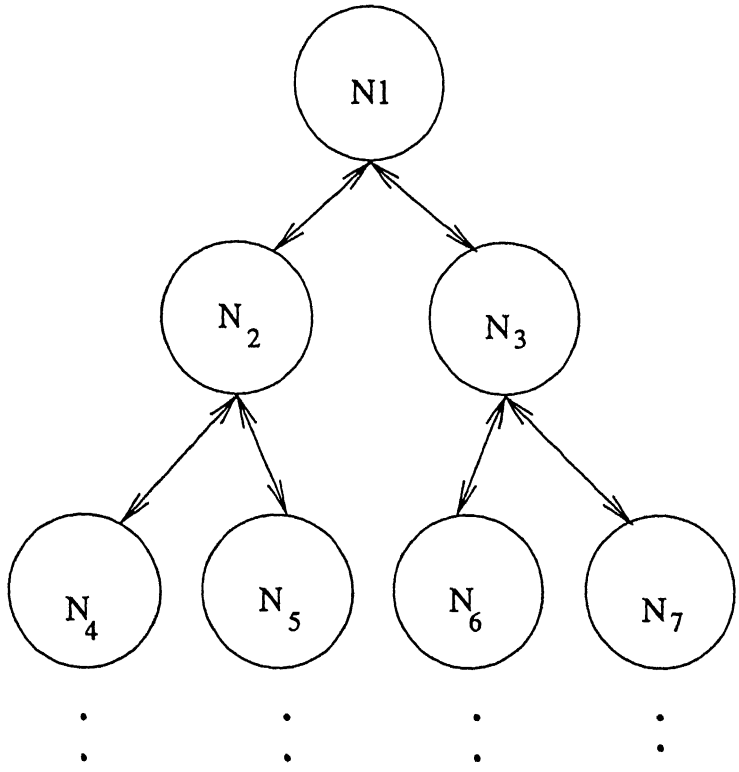


Figure 9: Operation of the AKS Network.

The network operates almost uniformly in this manner from start to finish. At any time the number of elements at each node at a particular depth of the tree is the same and this number increases in a geometric progression with depth from the root. The upper smaller sized nodes are concerned with recycling that small fraction of the elements which may have been misclassified in some partitioning process. As the time progresses, the number of elements at each node is reduced again in geometric progression, thus squeezing the elements down the tree towards their final locations in the leaves. The correctness of the network is demonstrated by proving an invariant which bounds the number of "wrong" elements at each node. AKS showed that for a fixed constant c, this invariant reduces to a value less than unity at a depth of $\lg n - c$ of the tree. This means that, at this point each element is in its proper leaf of the tree and what remains is to sort the 2^c elements in each node, which in turn can be done in $O(1)$ time. Further, processing at different nodes

of the same depth are disjoint. Hence, each level of the tree require $O(1)$ processing time (as each node require $O(1)$ time). Summing up, we see that to process up to a depth of $\lg n - c$, one needs $O(\log n)$ time, and from then, complete sorting is achieved in another $O(1)$ time. thus, the total delay gets bounded by $O(\log n)$.

Despite being $O(\log n)$, the AKS network unfortunately is of no practical value since the constants hidden in the Big-Oh notation are enormous. Even with the improvements on the network suggested in [9] the delay is well over $6000 \lg n$ and for all practical purposes, Batcher's network is much better. Further attempts to reduce the depth have come across very little success to this day.

On the other hand, the huge gap between the information theoretic lower bound of $2 \lg n - 2 \lg e - o(1)$ and the upper bound calls for improving the lower bound and reduce the disparity existing between the upper and lower bounds of the problem. However, the discussion in the next session shows that even this seems to be unexpectedly hard.

2.3 Lower Bounds

We have already seen that information theoretically one need $\Omega(\lg n!)$ comparisons for sorting n items which yields a lower bound of $n \lg n - n \lg e$ on size and $2 \lg n - 2 \lg e$ on depth. Simple improvements were made by Van Voorhis in 1972 who first derived a simple size bound of $n \lg n$ [11] and a later a more complicated bound of $n \lg n + \frac{1}{2} n \lg \lg n + \Omega(n)$ [12]. However, since then till 1980 no substantial improvements could be made on the depth bound while the size bound stayed for over 20 years.

A.C Yao (1980) proved the following results which we state without proof ([8],Theorem 4.5, corollary 4.6, p 576).

Theorem: Let $\alpha = 1/(3(2 - \lg 3))$. Then, for any fixed $\epsilon > 0$ there exists a number n_0 such that for all $n \geq n_0$,

$T(\lceil n^{\alpha} \rceil, n) \geq (3\alpha - \epsilon) \lg n \approx (2.41 - \epsilon) \lg n$.

corollary: There exists a number n_0 for every $\epsilon > 0$ such that $T(t, n) \geq (3\alpha - \epsilon) \lg n \approx (2.41 - \epsilon) \lg n$ for every $n \geq n_0$ and $n/2 \geq t \geq n^{\alpha}$.

Since an n-sorter is an (n,t)selector for all $1 \leq t \leq n$, it follows that the above bound also applies to sorting networks, thereby yielding a lower bound of $(2.41 - o(1))\lg n$ for depth for sufficiently large n. Observe that this result historically precedes AKS theorem. At that time, this bound was considered weak as it was generally believed that sorting is impossible in $O(\log n)$ time.

However, after the discovery of AKS network, several attempts at reducing the size of the constants associated with the network failed. Then, efforts were directed towards improving Yao's lower bound to reduce the mismatch between the upper and lower bounds. However, so far these efforts have met with little success. The only slight improvement so far was made by Nabil Kahale et al. [10] who proved a lower bound of $(1.12-o(1))n\lg n$ on size and $(3.27-o(1))\lg n$ on depth. The size bound is obtained by an interesting application of probabilistic method. They arrived at the size bound by counting the expected number of $0-1$ comparisons (called collisions in their terminology) required when a random n-vector from $\{0,1\}^n$ is given as input to an n-sorter. The depth bound is an improvement on Yao's method.

In this thesis, we look at sorting networks combinatorially using a new technique of analysis. although this yields no new results, it yields a better insight to the properties of comparison networks in general. Further, the approach used here might be applied to other similar problems as well.

Chapter 3

Minimal Sets

This chapter presents our work on sorting networks. Here, we develop an equivalent alternate formulation for the problem of sorting with a comparison network, based upon the idea of *minimal sets* which we shall see shortly. The zero-one principle is implicitly used in this formulation. In the first section of this chapter we introduce the concept of a minimal set and associated formalism. The next two sections explain their relevance to sorting networks. This shall be followed by a discussion of a lower bound obtained through arguments based on minimal sets. The last section presents the conclusions and scope for future work.

3.1 Basic Techniques

In this section, we introduce the notions of the *string set associated with a wire* and *minimal string set associated with a wire* in a comparison network. Later, we shall see a reformulation of the sorting problem based on them. Before formal definitions, we give here an informal introduction to the ideas involved.

Consider an n-sorter with vectors $< x_1,x_2...x_n >$ and $< y_1,y_2...y_n >$ at input and output respectively. We have seen from the zero-one principle that it is sufficient to consider inputs from $\{0,1\}^n$ alone to discuss sorting properties. Hence, we assume all x_i, y_i to belong to $\{0,1\}$.

For later notational convenience, hereafter, we reverse the numbering of the

output lines of an n-sorter. i.e., the i^{th} output wire will be assumed to carry the i^{th} largest output [1]. The revised numbering scheme is shown in Figure 10. Here, x_i and y_i will denote values assumed by the i^{th} input and output lines respectively of the n-sorter under consideration.

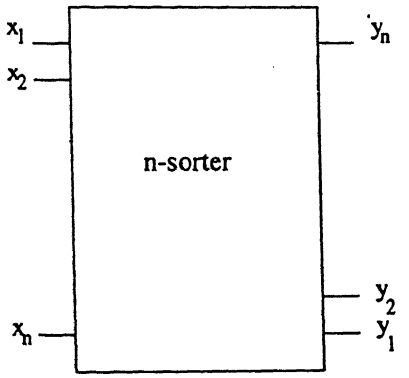


Figure 10: Revised ordering of output lines of n-sorter

We can think of the input vector x and the output vector y as strings from $\{0,1\}^n$. Hence, hereafter, we shall denote them as strings from $\{0,1\}^n$ and write them in the form $x = x_1x_2...x_n$ where each $x_i \in \{0,1\}$. Observe that when $x = x_1x_2...x_n$ is the input string to an n-sorter, the i^{th} input wire assumes value 1 when $x_i = 1$, irrespective of the values of $x_j, j \neq i$. The symbol $e^{(i)}$ shall be used to denote the string x with $x_i = 1$ and $x_j = 0$, $j \neq i$. Similarly $e^{(i)(j)...(r)}$ will denote the string $x = x_1x_2...x_n$ with $x_u = 1$ if $u \in \{i, j, ..., r\}$ and 0 otherwise. The *strength* or *order* of a string x denoted by $|x|$ is defined as the number of ones in the string. A string of order i may be conveniently called an *i-string*.

Now we can ask the question, what is the set of input strings for which the i^{th} input wire assume value 1. Evidently, there are 2^{n-1} such strings, since as long as the i^{th} position has value 1, we can arbitrarily give any value from $\{0,1\}$ to the remaining $n-1$ positions of the input string, still keeping the value at the i^{th} wire 1 when the string is applied as input to the network. Here, it is convenient to think

[1] Note that in the previous chapters we assumed that the i^{th} output line carries the i^{th} smallest output.

of the string $e^{(i)}$ as the "minimal" string required to give value 1 to the i^{th} input wire (minimal in the sense having least order). It is quite convenient to work with such "minimal" strings. Hence the following formalism is developed.

We introduce the partial order $\preceq$ on strings from $\{0,1\}^n$ as follows. For strings $x = x_1x_2...x_n$ and $y = y_1y_2...y_n$, we say $x \preceq y$ if for $1 \leq i \leq n$, $x_i \leq y_i$, where $\leq$ is the normal 'less than or equal to' relation between the numbers 0 and 1. If $x \preceq y$, we say $y \succeq x$. Note that the ordering is not total. Equality between strings is defined by $x = y$ if $x \preceq y$ and $y \preceq x$. We say $x \neq y$ if x and y are not equal. If neither $x \preceq y$ nor $y \preceq x$ we say x and y are *incomparable*. $x \wedge y$ shall stand for the string with the property $(x \wedge y)_i = 1$ if both $x_i = 1$ and $y_i = 1$, while $x \vee y$ shall denote the string with the property $(x \vee y)_i = 1$ if either $x_i = 1$ or $y_i = 1$. Clearly $x \wedge y \preceq x \preceq x \vee y$, $|x \wedge y| \leq |x| \leq |x \vee y|$ etc.

A set S of strings from $\{0,1\}^n$ is said to form a *minimal set* if for any $x, y \in S$ neither $x \preceq y$ nor $y \preceq x$. i.e., if every pair of strings in S are incomparable. We can make any set S of strings minimal by removing from S all $y \in S$ such that there exists some $x \in S$ satisfying the relation $x \preceq y$ and $x \neq y$. The new minimal subset obtained from S shall be denoted by $\mu(S)$. Clearly $\mu(S) = S$ iff S is minimal. The following fundamental result shows that $\mu(S)$ well defined.

Theorem (T1): The minimal subset $\mu(S)$ of a set S is unique.

Proof: Suppose X and Y be minimal subsets of S and let $X \neq Y$. Then, there should be some element x in one of them, say X such that $x \in X$ and $\neg(x \in Y)$. But $x \in S$. Hence, there must be some $y \in Y$ such that $y \preceq x$. But then, since $y \neq x$ and $y \preceq x$ we have $\neg(x \in X)$ which is a contradiction. **Q.E.D.**

As an example, consider the set $S = \{0101, 1010, 0111, 0011\}$ of strings from $\{0,1\}^4$. Here $0101 \preceq 0111$ and $0011 \preceq 0111$, other strings being incomparable. Hence $\mu(S) = \{0101, 1010, 0011\}$.

If S is the set of all strings from $\{0,1\}^n$ which when applied at the input of an n input comparison network, makes a particular wire l in the network assume value 1, we say S is the *string set associated with the wire* l and $\mu(S)$, the *minimal set associated with* l or simply the *minimal set of* l. The following observation follows directly from the definition of minimal set.

Lemma (L1): If S is the string set associated with a wire l in an n input comparison network and if for some particular input $x \in \{0,1\}^n$ l assumes value 1, then there must be some $y \in \mu(S)$ such that $y \preceq x$. conversely, if $x \in \{0,1\}^n$ and if there exists a $y \in \mu(S)$ such that $y \preceq x$, then l will assume value 1 when x is applied as input to the network.

Informally, the lemma states that any input which makes the wire l assume value 1 should have ones in positions corresponding to at least one of the strings in $\mu(S)$ and any input string that has ones in positions corresponding to at least one of the strings in $\mu(S)$ will make wire l assume value 1.

Clearly, the minimal set of the i^{th} input wire is $\{e^{(i)}\}$ for $1 \leq i \leq n$. We say a wire l is an *i-string wire* if the string with the lowest order in the minimal set of l has order i. The order of a minimal set is defined as the order of the string of minimal order in it. Thus, all the input wires to a comparison network are 1-string wires and their corresponding minimal sets are of order 1.

Let N be an n-sorter. Then, the i^{th} output wire of N will have value 1 if the input string to N has order $\geq i$. In particular, we observe that the minimal set of i^{th} output wire is the set of ${}^nC_i$ elements containing all strings of order i. We put down these observations formally for easy later reference.

Lemma (L2):

(a) The string set associated with the i^{th} input wire of an n-sorter N is $\{x \in \{0,1\}^n : e^{(i)} \preceq x\}$. The minimal set of the i^{th} input wire is $\{e^{(i)}\}$.

(b) The string set associated with the i^{th} output wire of an n-sorter is $\{y \in \{0,1\}^n : |y| \geq i\}$. The corresponding minimal set is $\{y \in \{0,1\}^n : |y| = i\}$.

Now, we shall move on to deduce the relation between the string sets associated with the input and output wires of a comparator.

3.2 Strings and Comparators

Consider a comparator C in an n-input comparison network with input wires a, b and output wires c, d, where c is the top output wire and d the bottom wire. Let S_a, S_b, S_c and S_d be the string sets associated with a, b, c and d respectively as shown in Figure

11. Our aim here is to find the relation between these sets and the corresponding minimal sets. The symbols defined above will be identically used throughout the remainder of this chapter with arbitrary comparators.

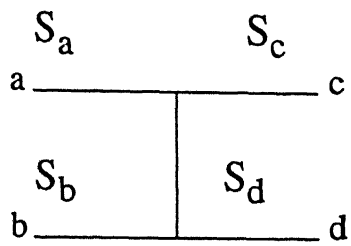


Figure 11: Minimal Sets in a comparator

The wire c will assume value 1 only if both a and b have value 1. Hence the strings associated with c are precisely the ones which are associated with both a and b. i.e., $S_c = S_a \cap S_b$. Similarly since d assumes value 1 when the values of either a or b is 1, we have $S_d = S_a \cup S_b$.

Now, consider the set $S_c' = \{x \vee y : x \in \mu(S_a) \text{ and } y \in \mu(S_b)\}$. Since $x \preceq x \vee y$ and $y \preceq x \vee y$, it follows from L1 that any string in S_c' when applied as input to the comparison network shall cause both a and b and hence c assume value 1. Similarly any string from the set $S_d' = \{x : x \in \mu(S_a) \text{ or } x \in \mu(S_b)\}$ will cause the wire d to assume value 1. We summarize these observations into the following lemma.

Lemma (L3):

(a)$S_c' \subseteq S_c = S_a \cap S_b$

(b)$S_d' \subseteq S_d = S_a \cup S_b$.

Now, we shall prove the following non-trivial result.

Theorem (T2):

(a) $\mu(S_c') = \mu(S_c)$

(b) $\mu(S_d') = \mu(S_d)$.

The theorem permits calculation of the minimal sets of the output wires of a comparator directly from the minimal sets of its input wires. Before proving it, we need the following results.

Lemma (L4): For any $y \in S_c$, there exists some $x \in S_c'$ such that $x \preceq y$. If $y \in S_d$,

then there exists some $x \in S'_d$ such that $x \preceq y$.

Proof: Suppose $y \in S_c$. Then, by L3, $y \in S_a$ and $y \in S_b$. Hence, by L1 there exists $x_1 \in \mu(S_a)$ and $x_2 \in \mu(S_b)$ such that $x_1 \preceq y$ and $x_2 \preceq y$. Therefore $x_1 \vee x_2 \preceq y$. But $(x_1 \vee x_2) \in S'_c$ by definition. Similarly, if $y \in S_d$, then there exists some $x \preceq y$ such that $x \in \mu(S_a)$ or $x \in \mu(S_b)$. In either case $x \in S'_d$. Hence proved.

Lemma (L5):

(a) $\mu(S_c) \subseteq S'_c$

(b) $\mu(S_d) \subseteq S'_d$

Proof: Let $x \in \mu(S_c)$. Then, $x \in S_c$. Hence by L4, there must be some $y \in S'_c$ such that $y \preceq x$. However, by definition of $\mu(S_c)$, $y \preceq x$ iff $y = x$. Hence, $x \in S'_c$. This proves (a). Proof for (b) is identical.

Now we are ready to prove T2.

Proof for T2: Using L5, we can write S'_c as the union of disjoint subsets $\mu(S_c) \cup C$ where $C = S'_c - \mu(S_c)$. If C is empty, $\mu(S'_c) = \mu(S_c)$ trivially. Otherwise, let $y \in C$. Since $y \in S_c$ (by L3), there must be some $x \in \mu(S_c)$ such that $x \preceq y$ (by L1). Hence $\neg(y \in \mu(S'_c))$ by definition of $\mu(S'_c)$. Thus, $\mu(S'_c) = \mu(S_c)$. This proves (a). Similarly (b) can be proved. **Q.E.D.**

We shall denote $\mu(S_c)$ as $\mu(S_a) * \mu(S_b)$ and $\mu(S_d)$ as $\mu(S_a) + \mu(S_b)$, defining $*$ and $+$ as binary operations on minimal string sets. We shall also use the terms **-wire* and *+-wire* to indicate the top and bottom output wires of a comparator. We say $x, y \in \{0,1\}^n$ are *components* of $z \in \{0,1\}^n$ if $z \neq x$, $z \neq y$ and $z = x \vee y$. Clearly every string in the minimal set of the *-wire of a comparator is either synthesized from a pair of component strings present in the minimal sets of the input wires of the comparator, or is itself present in the minimal set of one of the input wires.

Let us look at an example to illustrate theorem T2. Let $\mu(S_a) = \{010, 101\}$, $\mu(S_b) = \{110, 001\}$. Then $S'_c = \{110, 011, 111, 101\}$ and $S'_d = \{010, 101, 110, 001\}$. By T2, $\mu(S_c) = \mu(S'_c) = \{110, 101, 011\}$ and $\mu(S_d) = \mu(S'_d) = \{010, 001\}$. Here $(001) \in S_b$ and $(010) \in S_a$ are the components of $(011) \in \mu(S_c)$ and so on.

Before concluding this section, we shall prove the following result which has important consequences.

Lemma L6: If X and Y are minimal sets, then every $x \in X \cup Y$ is an element of

$Z = (X * Y) \cup (X + Y)$.

The lemma states that a minimal string associated with some wire at depth d of a comparison network will always be associated with some wire at depth $d+1$. i.e., a comparator cannot eliminate a minimal string from a comparison network.

Proof: Let $x \in X$. if there is no $y \in Y$ such that $y \preceq x$, then x is minimal in $X \cup Y$. i.e., $x \in \mu(X \cup Y)$ or $x \in X + Y$. Otherwise, if there exists some $y \in Y$ such that $y \preceq x$, then $x \vee y = x$ and hence $x \in S = \{x \vee y : x \in X \text{ and } y \in Y\}$. Further, since x is minimal in X, there is no other $x' \in X$ such that $x' \preceq x$. Also, for any $x' \in X$, $x' \preceq x' \vee y$ for all $y \in Y$. Hence, for no $x' \in X$ and $y \in Y$ it can be the case that $x' \vee y \preceq x$ unless $x = x'$ and $x' \vee y = x$. In any case, x is minimal in S. i.e., $x \in \mu(S)$ or $x \in X * Y$. The case for $x \in Y$ is symmetrical. Thus in all cases $x \in (X * Y) \cup (X + Y)$. Hence proved.

The arguments followed in the proof for L6 shows that for all $x \in X$ and $y \in Y$, if neither $x \preceq y$ nor $y \preceq x$, then every string in $X \cup Y$ will be in $X + Y$. Further in this case, all strings in $X * Y$ will be different from the strings in both X and Y and $X * Y$ will contain $|X||Y|$ strings. Thus we have:

Corollary C1: If X and Y are minimal sets, the following conditions are equivalent.

(a) For all $x \in X$ and $y \in Y$ neither $x \preceq y$ nor $y \preceq x$.

(b) $|X * Y| = |X||Y|$. i.e., every $x \vee y$ such that $x \in X$ and $y \in Y$ is in $X * Y$.

(c) $|X + Y| = |X| + |Y|$ i.e., $X \cup Y = X + Y$

(d) $X \cup Y$ and $X * Y$ are disjoint.

Now, suppose if it happens so that for all $x \in X$, $x \preceq y$ for some $y \in Y$, then $X * Y = Y$ and $X + Y = X$. In that case the comparator performs no real function and we say that the comparator is *redundant*. In this case, we can directly connect the input wire whose minimal set is X to the bottom output wire and the other input wire to the top output wire, thus eliminating the comparator. It may seem at this stage that such a shorting will lead to a cross connection of wires, and we will not be able to draw the comparison network containing the redundant comparator in the standard form after its removal. However, the discussion in [1] shows that even in this case we can redraw the network in the standard form without increasing the depth or size of the network.

Since for all $x, y \in \{0,1\}^n$, $|x| \leq |x \vee y|$, equality holding only when $x \vee y = x$, we add the following observation to C1.

Corollary C2: The order of $X * Y$ will be $\geq$ the order of both X and Y. Order of $X * Y$ will be greater than that of both X and Y iff any of the conditions listed in C1 holds.

With the theory developed so far, we shall try to look at some interesting properties of sorting networks.

3.3 Strings and Sorting Networks

Several simple properties of sorting networks can be discovered using analysis with minimal sets. A sample result is given below.

Proposition P1: If N is an n-sorter of depth d, then any comparator at depth d should connect adjacent lines or it is redundant.

Proof: Let C be a comparator at depth d of N. We shall follow the notation used in the previous section to denote the wires connected to C and the string sets associated with them. Thus S_a,S_b,S_c and S_d will denote the string sets associated with the input wires a,b and output wires c,d of C. Let d and c be the i^{th} and j^{th} output wires of N. Clearly $j > i$. By L2, $\mu(S_d) = \{x \in \{0,1\}^n : |x| = i\}$ is the set of all i-strings. Further, since $\mu(S_d) = \mu(S_a) + \mu(S_b)$ every string $x \in \mu(S_d)$ should belong either to $\mu(S_a)$ or to $\mu(S_b)$. No $x \in \mu(S_d)$ can belong to both $\mu(S_a)$ and $\mu(S_b)$ since in that case $x \in \mu(S_c)$ which is impossible since $j \neq i$. Thus, the i-strings in $\mu(S_d)$ are disjointly distributed between $\mu(S_a)$ and $\mu(S_b)$. We have two possibilities.

case 1: All i-strings belong to one of $\mu(S_a), \mu(S_b)$.

case 2: The i-strings are distributed between $\mu(S_a)$ and $\mu(S_b)$.

If case 1, Let $\mu(S_a)$ contain all i-strings. Then, every string in $\mu(S_b)$ must have order $> i$ or else that string will appear in $\mu(S_d)$ which is impossible since $\mu(S_d)$ has only i-strings. As c is the j^{th} output line, S_b should contain all j-strings, since $S_c = S_a \cap S_b$. Further, no r-string x for $j > r \geq i$ can occur in S_b as this will cause x to be in S_c since S_a also contain x (by L2,L3). Hence, $\mu(S_b)$ is the set of

all j-strings. However, in that case, we can directly connect a to d and b to c and eliminate C. Thus, in case 1, the C becomes redundant.

If case 2, i.e., if the i-strings are disjointly partitioned between $\mu(S_a)$ and $\mu(S_b)$, $\mu(S_d)$ must contain some $i+1$ strings. This follows from the fact that we cannot partition the set of all i-strings into non-empty subsets X and Y such that for all $x \in X$ and $y \in Y$ $|x \vee y| > i+1$.[2] Thus, $j \leq i+1$. But since $j > i$, it follows that $j = i+1$. Hence, P1 is proved.

Now we proceed to study the characteristics of the minimal sets associated with the wires at different levels of an n-sorter. We start with the following fundamental result.

Theorem T3: At a given depth d of a sorting network, a particular minimal string can be associated with at most one wire.

We need the following result to prove T3.

Lemma L7: If a minimal string x belongs to the minimal set of more than one wire at depth d of a comparison network, then x will belong to the minimal set of more than one wire at depth $d+1$.

Proof: Assume that a string x belongs to the minimal sets $\mu(X)$ and $\mu(Y)$ of wires w_1 and w_2 at depth d, where X and Y are the string sets associated with w_1 and w_2 respectively. Suppose that w_1 is compared with w_2 at depth d. Then, since $x \in X$ and $x \in Y$, $x \in X \cup Y$ and $x \in X \cap Y$. Further, since x is minimal to both X and Y, so should it be to $X \cup Y$ and $X \cap Y$. Hence, $x \in X + Y$ and $x \in X * Y$. Thus, more than one wire at depth $d+1$ has x in its minimal set.

The lemma follows vacuously when w_1 and w_2 are not connected to any comparator at depth d. Finally, L6 takes care of the case if either or both of them are compared with other wires at depth d. Hence proved.

Proof for T3: L2(b) states that every string in $\{0,1\}^n$ is associated with exactly one output wire of an n-sorter. hence, if any string x were minimal to more than one wire at any intermediate stage of an n-sorter, by L7, it should be associated with more than one wire at all subsequent stages including the output stage which

[2]The statement is equivalent to saying that the set of all strings of order i cannot be partitioned into two non-empty sets X and Y such that the hamming distance between any pair of strings $x \in X$ and $y \in Y$ is greater than two.

is impossible. **Q.E.D.**

As a corollary to this, we can modify L6 to the following form applicable to sorting networks.

corollary (Theorem T4): If X and Y are minimal sets associated with the input wires of any comparator in a sorting network, then every $x \in X \cup Y$ is an element of exactly one of either $X + Y$ or $X * Y$.

In corollary C1 of L6, we have seen the conditions in which, if X and Y are the minimal sets of the inputs wires to a comparator, $|X * Y| = |X|.|Y|$. If the comparator belongs to a sorting network, T3 implies that $X \cap Y = \emptyset$. i.e., no $x \in \{0,1\}^n$ can be an element of both X and Y. Now, we shall study the case when $|X * Y| < |X|.|Y|$.

Let $x \in X$ and $y \in Y$. In the proof L6, we have shown that if $x \leq y$, then $(y = x \vee y) \in X * Y$. Further, there can be no $(y' \neq y) \in Y$ such that $y' \preceq x$ as this would imply $y' \preceq y$ which is impossible. Hence x is minimal in $X \cup Y$ and thus $x \in X + Y$. Also, it is clear that $x \vee y = y$ only if $x \preceq y$. Hence, we have the following result.

Lemma L8: Let X and Y be the minimal sets associated with the input wires of a comparator in a sorting network. The following conditions hold.

(a) If $x \in X$ and $y \in Y$ and $x \preceq y$, then $y \in X * Y$ and $x \in X + Y$.

(b) If some $y \in Y$ is also an element of $X * Y$, then there must be some $x \in X$ such that $x \preceq y$ and $x \in X + Y$.

Note that the case for $y \preceq x$ is symmetrical.

We shall see one more result on the characteristics of strings in sorting networks.

Theorem T5: Let $x, y, z \in \{0,1\}^n$ such that $z = x \vee y$ with x and y incomparable. Also, let x, y, z belong to the minimal sets of some wires at depth d of an n-sorter. Then, for all depth $i \geq d$, if X_i, Y_i and Z_i are the minimal sets containing x, y and z respectively in the network, then $X_i \neq Y_i$ iff there exists either some $(y' \neq y) \in Y_i$ such that $y' \preceq x$ or some $(x' \neq x) \in X_i$ such that $x' \preceq y$.

Note that $X_i \neq Z_i \neq Y_i$ as otherwise z will not be minimal in the set to which it belongs to. The theorem puts considerable restriction over the distribution of strings in the minimal sets of various wires at each stage of a sorting network.

To prove this result, we make use of the fact that introduction of an additional comparator between any two wires at any stage of a sorting network still yields a sorting network.

Proof: For the purpose of contradiction, assume that X_i, Y_i and Z_i are the minimal sets associated with different wires w_1, w_2 and w_3 at some depth $i \geq d$ and for all $x \in X_i, y \in Y_i$, neither $x \preceq y$ nor $y \preceq x$. Now, if we introduce a new comparator between w_1 and w_2, then $(z = x \vee y) \in X * Y$ by corollary C1 of L6 and z will be associated with more than one wire at all subsequent stages of the network (by L7) which is impossible by T3. **Q.E.D.**

To illustrate the effect of T5, consider the 2-string $e^{(i)(j)}$. $e^{(i)}$ and $e^{(j)}$ are it's unique components and they are minimal elements of $\{\{0,1\}^n, \preceq\}$. Hence, once $e^{(i)(j)}$ is formed from $e^{(i)}$ and $e^{(j)}$, at all subsequent stages $e^{(i)}$ and $e^{(j)}$ will be in the minimal set of the same wire while $e^{(i)(j)}$ will be in the minimal set of some other wire.

Figure 12 shows the minimal sets associated with various wires of a 4-sorter.

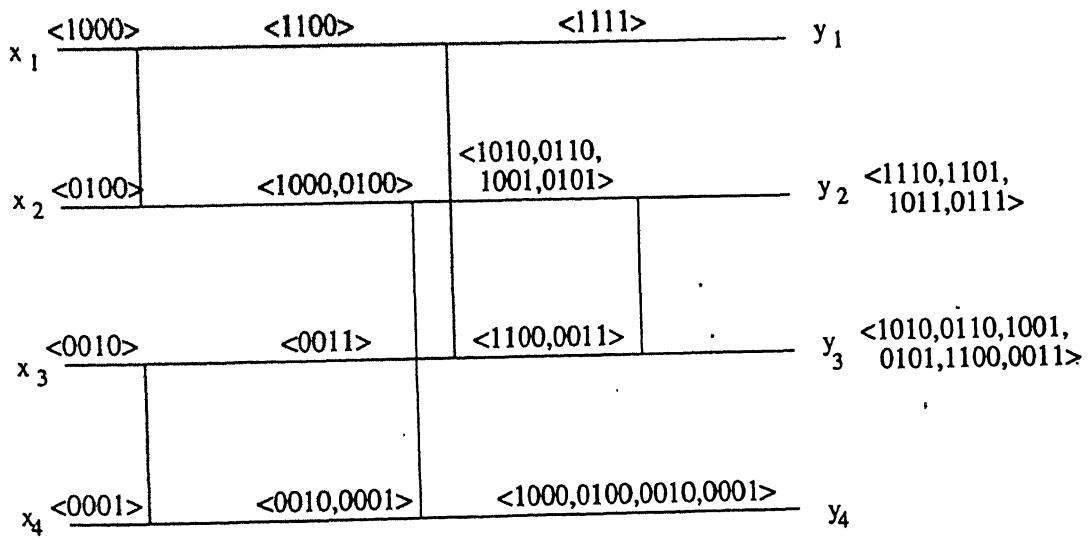


Figure 12: Minimal sets in a 4-sorter

T3,T4 and T5 depicts the essential features of minimal strings in a sorting network. We proceed now to apply the above facts to obtain a lower bound for the

sorting problem. However, we should note here that more intricate and interesting hidden properties might exist which could be revealed by a harder analysis.

One consequence of our discussion so far in this chapter is that **the problem of sorting using a comparison network can be translated into the problem of going from the set $\{e^{(i)}\}$ to $\{x \in \{0,1\}^n : |x| = i\}$ for all lines $1 \leq i \leq n$ in a comparison network**, of course, under the constraints imposed by T3,T4 and T5. In particular, **a lower bound obtained for the depth or size of a comparison network for the problem with strings will also be a lower bound for the sorting problem.** The next section presents a result in this direction. As it turned out, the problem is hard to be analyzed with straight combinatorial methods that we have employed here. We present this result with the hope of setting a platform for later improvements using more profound mathematical tools for analysis.

3.4 A Lower Bound

In this section we apply minimal sets to derive a simple lower bound for the size of an n-sorter. For this, we introduce a classification of minimal strings as follows.

- All 1-strings belong to class 0.
- All i-strings for $2^{p-1} < i \leq 2^p$ belong to class p for $1 \leq p \leq \lceil \lg n \rceil$.

A wire is said to belong to class p if the string of lowest order in it's minimal set belongs to class p.

Evidently, a string of class $p+1$ can be formed only at the *-wire of a comparator, at least one of whose input wires carry a string of class p or $p+1$. Hence, a wire of class $p+1$ can be formed only at the *-wire of a comparator with at least one of whose inputs being a wire of class p or $p+1$. Further, at the output stage of an n-sorter, we have one class 0 line and 2^{p-1} class p lines for $1 \leq p \leq \lfloor \lg n \rfloor$, with the remaining lines belonging to class $\lceil \lg n \rceil$.

From the discussion of minimal sets made in the preceding sections, we make the following observations.

- Comparison of two class-i wires at depth d result in a class-i wire and a wire belonging to either class-i or class-$i+1$ wire at depth $d+1$.

- Comparison of a class-i wire with a class-j wire, $i < j$, at depth d gives rise to a class-i wire along with a wire of class either j or $j+1$ at depth $d+1$.

Now suppose at depth d of an n-sorter we have r class-i lines. Then each comparator placed between two class-i lines will reduce the number of class-i lines at depth $d+1$ at most by 1. In general, if we have r class-i lines in an n-sorter and if we want to reduce their number to t ($t < r$), we need $r-t$ comparators and depth at least $(r-t)/(\frac{n}{2}) = 2(r-t)/n$ since at most $n/2$ comparators may be there at any depth of an n-sorter.

We know that at the output stage of an n-sorter, there is one class-0 wire and exactly 2^{p-1} class-p wires for $1 \leq p \leq \lfloor \lg n \rfloor$. We have also seen that a wire of class $p+1$ cannot be formed without having a class p wire. These observations lead us to the following argument.

Consider an n-sorter. At it's input stage, there are n lines of class 0. To converge them to one line at the output, we need to use $n-1$ comparators. In this process, we generate $n-1$ class-1 wires at different stages of the network. To converge them to one wire at the output, we need another $n-2$ comparators yielding $n-2$ class-2 wires. In general, for all $1 \leq p < \lfloor \lg n \rfloor$, we will have $n - 2^{p-1}$ class p lines and then use $n - 2^p$ comparators to merge them to 2^{p-1} lines at the output. Hence, there should be at least $\sum_{i=0}^{\lfloor \lg n \rfloor - 1}(n - 2^i) = n\lfloor \lg n \rfloor - 2^{\lfloor \lg n \rfloor} + 1 \geq n\lfloor \lg n \rfloor - n + 1$ comparators in any sorting network. Further, this requires at least a depth of $(n\lfloor \lg n \rfloor - n + 1)\,/\,(n/2) = 2\lfloor \lg n \rfloor - 2 + O(\frac{1}{n})$. The last term will have the effect of increasing the depth by unity since depth and size values must be integral. Hence, we have

Theorem T6:

$S(n) \geq n\lfloor \lg n \rfloor - n + 1$

$T(n) \geq 2\lfloor \lg n \rfloor - 1.$

The above results are weaker than the best known lower bounds in the literature, however, they are better than the ones obtained by information theoretic arguments.

Note that by working with string classes, we have left out the effects of minimal strings within a class in increasing the depth and size of a sorting network. We have done this simplification because without it analysis becomes too complex to be tackled with direct combinatorial methods and we have not reached at a method to tackle the complexity of such an analysis. We hope research in future will come up with tools to solve this problem.

We conclude this exposition with the following section which summarizes what we have seen so far and suggests aspects for future work.

3.5 Suggestions for Future Work

In the preceding sections of this chapter, we have seen some simple facts about sorting networks viewed in terms of minimal sets associated with it's wires. We have also seen a lower bound arising out of these facts. Also we have discussed possible improvements which could be carried out to yield stronger results. Further, most of the results we have seen here are about sorting networks and it will be interesting to investigate how much of them extend to general comparison networks.

We hope that the approach used in studying sorting networks here shall be useful to handle several problems in general comparison networks, communication networks, switching circuits etc. The essential idea here is to associate with each wire in a network, the set of inputs which, when applied to the network gives the wire a particular value, and then study the properties of the whole network in terms of these sets. This approach should be useful in those cases where we are able to calculate the sets associated with the wires at later stages of the network from the known sets of the initial stages as in the case of comparison networks (Theorem T2). As it turns out, comparison networks form just one platform in which the idea finds a realization.

Bibliography

[1] D. E. Knuth. *The Art of Computer Programming. Volume 3: Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.

[2] T. H. Kormen, C. E. Lisserson, R. L.Rivest, *Introduction to Algorithms* MIT Press, McGraw-Hill, 1994.

[3] N. Pippenger. Communication Networks. In J. van Leeuwen, editor, *Handbook of Theoretical computer Science, Volume A: Algorithms and Complexity*, pages 805-833. North-Holland, Amsterdam, 1990.

[4] M. Ajtai, J. Komlos, and E. Szemeredi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1-19, 1983.

[5] M. Ajtai, J. Komlos, and E. Szemeredi. An $O(n \log n)$ sorting network. In *Proc. 15th Ann. ACM Symp. on Theory of Computing*, pages 132-140, 1983.

[6] A. C. Yao and F. F. Yao. Lower bounds on merging networks. *J. Assoc. Comput. Mach.*, 23:423-432, 1976.

[7] A. C. Yao. Bounds on selection networks. *SIAM J. Comput.*, 9:566-582, 1980.

[8] K. E. Batcher, Sorting Networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 32, pages 307-314, 1968.

[9] M. S. Paterson. Improved sorting networks with $O(\log N)$ depth. *Algorithmica*, 5:75-92, 1990.

[10] Lower bounds N. Kahale et al. Lower bounds for sorting networks. *Proceedings of the ACM symposium on Theory of Computing*, volume 2, pages 437-445, 1995.

[11] D. C. Van Voorhis. Toward a lower bound for sorting networks. In R. E. Miller and J. W. Thatcher, editors, *The Complexity of Computer Computations*, pages119-129. Plenum Press, New York, NY, 1972.

[12] D. C. Van Voorhis. An improved lower bound for sorting networks. *IEEE Trans. Comput.*, 21:612-613, 1972.